图文 067、阶段性复习：JVM性能优化到底该怎么做?

807 人次阅读　2019-09-05 07:00:00

详情　评论

阶段性复习：
JVM性能优化到底该怎么做?

狸猫技术窝专栏上新，基于**真实订单系统**的消息中间件（mq）实战，重磅推荐：

未来3个月，我的好朋友原子弹大侠将带你一起，全程实战，360度死磕MQ

（点击下方蓝字进行试听）

从 0 开始带你成为消息中间件实战高手

重要说明：

如何提问：每篇文章都有评论区，大家可以尽情在评论区留言提问，我都会逐一答疑

（**ps**：评论区还精选了一些小伙伴对**专栏每日思考题的作答**，有的答案真的非常好！大家可以通过看别人的思路，启发一下自己，从而加深理解）

如何加群：购买了狸猫技术窝专栏的小伙伴都可以加入**狸猫技术交流群**。

（群里有不少**一二线互联网大厂的助教**，大家可以一起讨论交流各种技术）

具体**加群方式**请参见文末。

（**注**：以前通过其他专栏加过群的同学就不要重复加了）

狸猫技术

进店逛

相关频道

从 0 开
战高手
已更新1

0、写在前面的话

最近有个别同学后台留言问为什么最近的文章的一些顺序和标题，跟大纲里的有点出入，我在这里解释一下。

因为在实际写作的过程中，会把一些文章的顺序做一些交换，比如之前把Week 10里的内容放到了Week 9。

同时Week 9里有一些案例，在写作过程中考虑了一下，感觉有些东西没太大必要了，所以替换成了别的几个案例放在了Week 10里，因此有一些案例的标题上会有一些变更。

另外Week 10里把原定的3个案例替换为了3讲内容用来进行阶段性的复习，这个也是因为发现很多同学对我们专栏里的阶段性定时复习，定时停下脚步做点总结和梳理，非常的好评和认可，因此我们临时在这里加入了3讲内容进行阶段性梳理、总结和复习。

相信大部分人都认可一个观点：学习不是不停的学习新的内容，更重要的是在学新内容的同时，要经常性的停下脚步来复习之前的内容，这样反复的周期性的复习和总结，不断的重复和强化一些核心的东西，最后在三四个月的专栏学习完毕之后，才能彻底吃透和消化掉这块知识。

这就是我对原定大纲做出一些顺序变换以及少数案例替换为阶段性复习的原因和说明，希望大家理解我的良苦用心，好好学新内容，也好好复习老内容，消化成自己的东西，最后才真正有所收获。

1、一个新系统开发完毕之后如何设置JVM参数?

之前花费了很多的精力给大家介绍，在一个新系统开发完毕之后，到底该如何预估性的合理设置JVM参数?

毕竟直接用默认的JVM参数部署上线再观察，是非常的不靠谱的。很多公司也没有所谓的JVM参数模板。

首先大家应该估算一下自己负责的系统每个核心接口每秒多少次请求，每次请求会创建多少个对象，每个对象大概多大，每秒钟会使用多少内存空间?

这样接着就可以估算出来Eden区大概多长时间会占满?

然后就可以估算出来多长时间会发生一次Young GC，而且可以估算一下发生Young GC的时候，会有多少对象存活下来，会有多少对象升入老年代里，老年代对象增长的速率大概是多少，多久之后会触发一次Full GC。

通过一连串的估算，就可以合理的分配年轻代和老年代的空间，还有Eden和Survivor的空间

原则就是：尽可能让每次Young GC后存活对象远远小于Survivor区域，避免对象频繁进入老年代触发Full GC。

最理想的状态下，就是系统几乎不发生Full GC，老年代应该就是稳定占用一定的空间，就是那些长期存活的对象在躲过15次Young GC后升入老年代自然占用的。然后平时主要就是几分钟发生一次Young GC，耗时几毫秒。

2、在压测之后合理调整JVM参数

任何一个新系统上线都得进行压测，此时在模拟线上压力的场景下，可以用jstat等工具去观察JVM的运行内存模型：

Eden区的对象增长速率多块?
Young GC频率多高?
一次Young GC多长耗时?
Young GC过后多少对象存活?
老年代的对象增长速率多高?
Full GC频率多高?
一次Full GC耗时?

压测时可以完全精准的通过jstat观察出来上述JVM运行指标，让我们对JVM运行时的情况了如指掌。然后就可以尽可能的优化JVM的内存分配，尽量避免对象频繁进入老年代，尽量让系统仅仅有Young GC。

3、线上系统的监控和优化

系统上线之后，务必进行一定的监控，高大上的做法就是通过Zabbix、Open-Falcon之类的工具来监控机器和JVM的运行，频繁Full GC就要报警。

比较差一点的做法，就是在机器上运行jstat，让其把监控信息写入一个文件，每天定时检查一下看一看。

一旦发现频繁Full GC的情况就要进行优化，优化的核心思路是类似的：通过jstat分析出来系统的JVM运行指标，找到Full GC的核心问题，然后优化一下JVM的参数，尽量让对象别进入老年代，减少Full GC的频率。

4、线上频繁Full GC的几种表现

其实通过之前的各种案例，大家可以总结出来，一旦系统发生频繁Full GC，大概看到的一些表象如下：

机器CPU负载过高；
频繁Full GC报警；
系统无法处理请求或者处理过慢

所以一旦发生上述几个情况，大家第一时间得想到是不是发生了频繁Full GC。

5、频繁Full GC的几种常见原因

之前给大家分析过多个案例，通过那些案例的总结和归纳，可以得出下面几个常见的频繁Full GC的原因：

系统承载高并发请求，或者处理数据量过大，导致Young GC很频繁，而且每次Young GC过后存活对象太多，内存分配不合理，Survivor区域过小，导致对象频繁进入老年代，频繁触发Full GC。

系统一次性加载过多数据进内存，搞出来很多大对象，导致频繁有大对象进入老年代，必然频繁触发Full GC

系统发生了内存泄漏，莫名其妙创建大量的对象，始终无法回收，一直占用在老年代里，必然频繁触发Full GC

Metaspace（永久代）因为加载类过多触发Full GC

误调用System.gc()触发Full GC

其实常见的频繁Full GC原因无非就上述那几种，所以大家在线上处理Full GC的时候，就从这几个角度入手去分析即可，核心利器就是jstat。

如果jstat分析发现Full GC原因是第一种，那么就合理分配内存，调大Survivor区域即可。

如果jstat分析发现是第二种或第三种原因，也就是老年代一直有大量对象无法回收掉，年轻代升入老年代的对象病不多，那么就dump出来内存快照，然后用MAT工具进行分析即可

通过分析，找出来什么对象占用内存过多，然后通过一些对象的引用和线程执行堆栈的分析，找到哪块代码弄出来那么多的对象的。接着优化代码即可。

如果jstat分析发现内存使用不多，还频繁触发Full GC，必然是第四种和第五种，此时对应的进行优化即可。

6、一个统一的JVM参数模板

为了简化JVM的参数设置和优化，建议各个公司和团队leader做一份JVM参数模板出来，设置一些常见参数即可

核心就是一些内存区域的分配，垃圾回收器的指定，CMS性能优化的一些参数（比如压缩、并发，等等），常见的一些参数，包括禁止System.gc()，打印出来GC日志，等等。

这些常见的参数，之前基本都讲过了，建议大家自行整理出来一份模板即可。

End

如何加群?

添加微信号：Lvgu0715_ （微信名：绿小九），狸猫技术窝管理员

发送 Jvm专栏的购买截图

由于是人工操作，发送截图后请耐心等待被拉群

最后再次提醒：通过其他专栏加过群的同学，就不要重复加了

狸猫技术窝其他精品专栏推荐:

21天互联网java进阶面试训练营（分布式篇）